# Patchlayer: No confíes en el parche, sino en la evidencia

Compuerta de evidencia para parches de seguridad generados por IA

**Equipo:** David Briceño, Daniel Barrera, Santiago Ramos, Samuel Noreña

Investigadores independientes

**Track:** Automated Cyberdefense

| Repositorio | Demo | Contacto |
|---|---|---|
| `https://github.com/samuelnor/patchlayer` | Ejemplo de expediente | `s.norenat@uniandes.edu.co` |

**Resumen**

Los asistentes de código pueden generar parches que pasan pruebas y bloquean el exploit conocido, pero dejan vulnerabilidades explotables. Patchlayer es una compuerta para integración continua que evalúa cada parche con un contrato humano independiente: pruebas funcionales, exploit y análisis simbólico con CrossHair/Z3. En 25 parches sobre cinco casos CWE-20 y CWE-639, la línea base aceptó 10 de 15 parches superficiales; Patchlayer aceptó cero, sin rechazar parches correctos.

## 1. La amenaza

Cuando un LLM propone corregir una vulnerabilidad, los equipos suelen aceptar el parche si pasa las pruebas existentes y bloquea el exploit reportado. Pero una suite cubre solo casos previstos: el modelo puede sobreajustarse y añadir filtros ad hoc que detienen esa entrada, pero dejan la falla explotable ante variaciones. El resultado es una falsa sensación de seguridad para desarrolladores y usuarios [1, 2].

## 2. Lo que construimos

Patchlayer no repara código: determina si hay evidencia suficiente para integrar un parche a `main`. Antes de evaluar, el equipo redacta un contrato humano inmutable, sellado con SHA-256, que reúne pruebas funcionales (`test_funcional.py`), un exploit conocido (`test_exploit.py`) y una propiedad formal sobre funciones puras (`propiedad.py`). Patchlayer se ejecuta en integración continua o por CLI: recibe el caso vulnerable y el parche candidato, los copia a un directorio temporal y carga el objetivo en un subproceso limpio. Luego ejecuta las pruebas funcionales y de exploit, y verifica la propiedad con CrossHair/Z3 [3]. Un fallo rechaza el parche; un error o timeout lo deja no concluyente; solo aprobar todas las capas genera evidencia favorable. El expediente JSON registra hashes, tiempos, resultados por capa, contraejemplos y supuestos no verificados.

## 3. Cómo lo evaluamos

Probamos 25 pares (caso, parche) en cinco casos de dos familias: validación numérica (CWE-20) y autorización IDOR (CWE-639). El corpus contiene 15 parches superficiales (cinco controles vulnerables y diez filtros de lista negra), cinco regresivos y cinco correctos. Comparamos A




Figura 1: El contrato independiente somete cada parche a pruebas funcionales, exploit conocido y verificación simbólica; la compuerta aplica fallo cerrado y emite un expediente auditable.

(pruebas funcionales), B (funcionales + exploit) y C (cadena completa con CrossHair/Z3). Medimos la aceptación de parches superficiales (falsos negativos) y correctos (falsos rechazos), indicadores de seguridad y utilidad.

## 4. Resultados

B bloqueó los cinco controles vulnerables, pero aceptó 10 de 15 parches superficiales (66,7 %); C aceptó cero. La verificación simbólica añadió diez rechazos sin falsos rechazos: B y C aceptaron los cinco parches correctos. Las tres configuraciones rechazaron los cinco regresivos. La Tabla 1 resume los resultados.

Cuadro 1: Aceptación de parches por configuración, sobre 25 parches en cinco casos CWE-20 y CWE-639.

| **Métrica** | **Línea base B** | **Patchlayer C** | **Diferencia** |
|---|---|---|---|
| Superficiales aceptados (n=15) | 10/15 (66,7 %) | 0/15 (0 %) | -10 casos |
| Regresivos aceptados (n=5) | 0/5 | 0/5 | 0 |
| Correctos aceptados (n=5) | 5/5 | 5/5 | 0 |

## 5. Límites y doble uso

El análisis simbólico es acotado: cubre funciones puras de Python, no prueba universalmente el programa ni detecta efectos colaterales o propiedades ausentes del contrato. Los contraejemplos pueden convertirse en exploits contra el código vulnerable, y ejecutar parches generados también implica riesgo. Por ello, la evaluación debe aislarse en una sandbox; el expediente registra límites y supuestos no verificados. No probamos otros lenguajes ni sistemas con estado.

## Referencias

[1] Smith et al. (2015). *Is the cure worse than the disease?* ESEC/FSE.

[2] Qi et al. (2015). *An analysis of patch plausibility and correctness.* ISSTA.

[3] CrossHair Team (2026). *CrossHair documentation.*